Comparison of Heapsort, Quicksort, dualPivotQuickSort and Introsort

Başak ER
Department of Computer Science Dokuz Eylul University
Buca, IZMIR, 35390
basak.er@ceng.deu.edu.tr

April 7, 2019

ABSTRACT

Sorting is a fundamental operation in computer science (many programs use it as an intermediate step). A large number of sorting algorithms have been made in order to have a best performance in terms of computational complexity (best, average and worst), memory usage, stability and method. One of sorting algorithms can perform much better than another. For that, I intend to make a comparison paper and four representative algorithms were chosen to show these differences. In the paper, a comparative performance evaluation of four sorting algorithms Heap-Sort, Quick-Sort, dualPivotQuick-Sort and Intro-Sort will be presented depending on the performance factors shown above. Furthermore, the use of these algorithms will be discussed.

1 Introduction

Sorting is a fundamental operation in computer science. Sorting generally refers to the operation of ordering data in a given sequence such as increasing sequence or decreasing sequence. There are many fundamental and advance sorting algorithms. All sorting algorithm are problem oriented means they work well on some special problem and do not work well for any kind of a problems. Sorting algorithms are applied on specific kind of problems. Sorting algorithms are used for small number of elements, some sorting algorithms are used for large numbers, some sorting algorithms are used for floating number of data, and some are used for repeated values in a list. We sort data in numerical order or alphabetical order, arranging the list either in increasing order or decreasing order and alphabetical value like addressee key.

In this paper I discuss four sorting algorithms which are already exist named as Heap Sort , Quick Sort, Introsort and we design a new sorting algorithm named as dualPivotQuick Sort. In this paper I check the performance and comparison of all four-sorting algorithm on the basis of increasing the no of elements in bulk.

I check how much processing time is taken by all three sorting algorithms with dualPivotQuick Sort and compared them and finding which sorting algorithm takes less time to sort the elements like 1000, 10000, 100000. I also compared these algorithms with the equal, increasing, decreasing and random occurrences of the elements in the stack. If any algorithm takes less processing time it means that it sorts the element faster than others.

2 Sorting Algorithms

2.1 Heap Sort

Heapsort is a comparison-based sorting algorithm to create a sorted array (or list). The idea is to look at the array to sort as it was a binary tree. The first element is the root, the second is descending from the first element. The aim is to obtain a heap, thus a binary tree verifying the following properties: The maximal difference between the two leaves is 1 (all the leaves are on the last, or on the penultimate line), the leaves having the maximum depth are ―heaped‖ on the left, each node has a bigger value than its of its two children, for an ascending sort.

Heapsort is a much more efficient version of selection sort. HeapSort is not stable, that means heapsort may change the relative order of records with equal keys. HeapSort is in-place algorithm (don't need an auxiliary array). The complexity of the heap-sort algorithm, for sorting a n elements array, is $O(\, n \log_2 n)$.

2.2 Quick Sort

Quicksort is a divide and conquer algorithm which relies on a partition operation: to partition an array an element called a pivot is selected. All elements smaller than the pivot are moved before it and all greater elements are moved after it. This can be done efficiently in linear time and in-place. The lesser and greater sublists are then recursively sorted. Efficient implementations of quicksort (with in-place partitioning) are typically unstable sorts and somewhat complex, but are among the fastest sorting algorithms in practice.

Together with its modest O(log n) space usage, quicksort is one of the most popular sorting algorithms and is available in many standard programming libraries. The most complex issue in quicksort is choosing a good pivot element; consistently poor choices of pivots can result in drastically slower $O(n^2)$ performance, if at each step the median is chosen as the pivot then the algorithm works in O(n log n). Finding the median however, is an O(n) operation on unsorted lists and therefore exacts its own penalty with sorting.

In this paper, I did the pivoting process in Quick Sort in three different ways. The first method we call “FirstElement” is that the pivot is always the first element of the array to be sorted. The second one is called “RandomElement”, the pivot has any element in the array to be sorted. Finally, the third one is called “MidOfFirstMidLastElement, the pivot is an element whose value is in the middle among the {first, middle, last} elements in the array to be sorted.

2.3 Dual Pivot Quick Sort

In the dualPivotQuickSort method, I implemented a different version of quicksort. This version has two pivots: one in the left end and one in the right end of the array. The left pivot (LP) always less than or equal to the right pivot (RP).

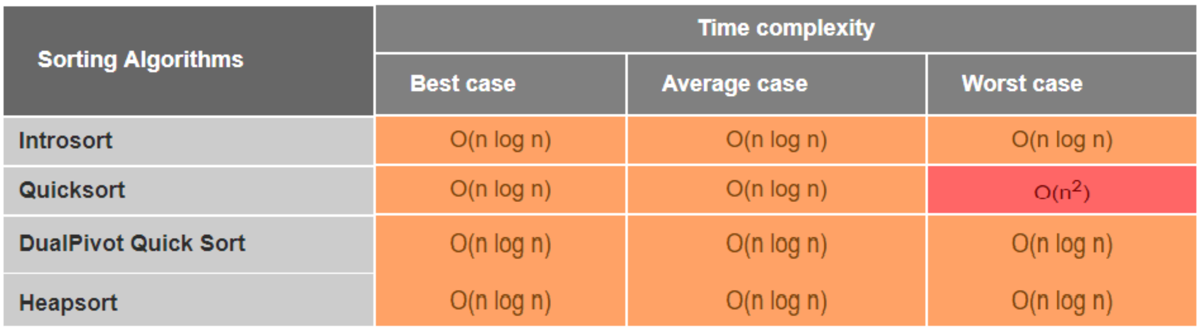
Input array is divided into three parts. In the first part, all elements will be less than LP, in the second part all elements will be greater or equal to LP and also will be less than or equal to RP, and in the third part all elements will be greater than RP. Until the existing pivots have stopped their moving, sorting process continues recursively.

2.4 Intro Sort

Introsort is an efficient in-place sorting algorithm, which usually beats all other sorting algorithms in terms of performance. Due to its high performance, it is used in a number of standard library sort functions. Introsort is a comparison sort, meaning that it can sort items of any type for which a less-than relation is defined. It is usually not a stable sort which means that if two elements have the same value, they might not hold the same relative position in the output as they did in the input.

Introsort is hybrid of quicksort and heapsort algorithm. A hybrid algorithm combines two or more other algorithms that solve the same problem, so that the resultant algorithm is better than the individual algorithms. Introsort begins with quickSort and switches to heapSort when the recursion depth exceeds a level based on (the logarithm of) the number of elements being sorted. When the number of elements is below some threshold, Introsort can switch to a non-recursive sorting algorithm such as insertion sort that performs fewer swaps, comparisons or other operations on such small arrays.

3 Algorithms and Big-O representations



Sorting Algorithms	Time complexity		
	Best case	Average case	Worst case
Introsort	O(n log n)	O(n log n)	O(n log n)
Quicksort	O(n log n)	O(n log n)	$O(n^2)$
DualPivot Quick Sort	O(n log n)	O(n log n)	O(n log n)
Heapsort	O(n log n)	O(n log n)	O(n log n)

Table 3.1

4 Performance Analysis

Heap Sort, Quick Sort, Dual Pivot Quick Sort, Intro Sort I applied in Eclipse and I test the random, equal, decreasing and increasing sequence input of length 1000, 10000, 100000 to check the performance. All the four sorting algorithms were executed on machine with 64-bit Operating System having Intel(R) Core i7-7700HQ processor @ 2.8 GHz, 2.4 GHz and installed memory (RAM) 16.00 GB. The times taken by the CPU at execution for Different inputs are shown in the table (Table 4.1). The length of input and CPU time taken (n sec) is shown in figure.

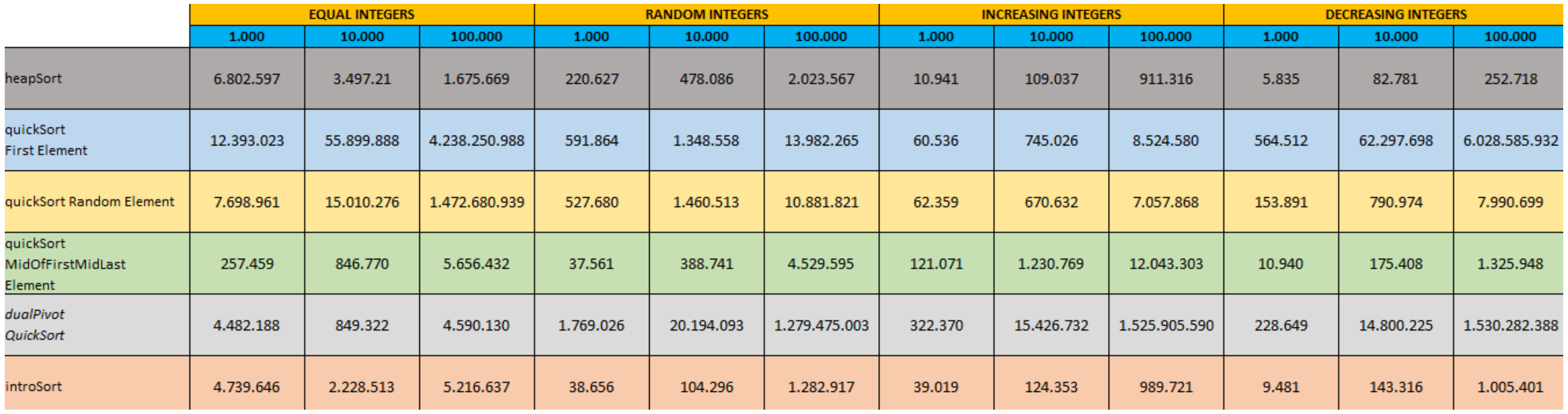
Result shows that Heapsort is typically somewhat slower than Quicksort, but the worst-case running time is always Θ(nlogn). Quicksort is usually faster, though there remains the chance of worst-case performance except in the Introsort variant, which switches to Heapsort when a bad case is detected. If it is known in advance that Heapsort is going to be necessary, using it directly will be faster than waiting for Introsort to switch to it. If you want to predict precisely when your algorithm will have finished, it is better to use the Heapsort algorithm, because the complexity of this one is always the same.

It is also the case if you know in advance that your array is almost sorted. QuickSort turns into a bad algorithm in the worst case. As in the reverse order. Because the worst case is n squared, it runs very slowly in descending arrays. Intro sort works very well in close numbers. As the difference between the numbers increases, it loses its effectiveness. Again, in the Introsort, random numbers are slower than others. Because the depth is equal to or less than the number of Heapsort is engaged.

In Dual Pivot Quick Sort, if a bad pivot is selected, it can be imagined that the her less her subset is always empty. This means that we only create a smaller subset of items each time, which, in the worst case, gives us O (n^2) behavior. If you select the first item, it can be the smallest item in the listed list and give the worst-case behavior. You can select a random item or the median of the three (front, center, end). Quicksort is fast because it uses a spatial location - by entering neighboring elements, comparing them with the pivot value (stored in a registry). Uses caching very effectively.

The pivot is often shifted forward, so it remains out of the way during pivoting. Then, it is placed in place (the pivot is maintained with a pivot element equal to or smaller than it). Quicksort algorithm is complex and needs to pass left and right boundary variables.

In addition, I think we should use heapsort to rank the popularity of the tweets containing hashtags in a Java hasp map. Because it is faster to exchange numbers close to each other. When we think of the numbers as the tree structure, there will be more nodes nearby. Therefore, less exchange will occur.



	EQUAL INTEGERS			RANDOM INTEGERS			INCREASING INTEGERS			DECREASING INTEGERS		
	1.000	10.000	100.000	1.000	10.000	100.000	1.000	10.000	100.000	1.000	10.000	100.000
heapSort	6.802.597	3.497.21	1.675.669	220.627	478.086	2.023.567	10.941	109.037	911.316	5.835	82.781	252.718
quickSort First Element	12.393.023	55.899.888	4.238.250.988	591.864	1.348.558	13.982.265	60.536	745.026	8.524.580	564.512	62.297.698	6.028.585.932
quickSort Random Element	7.698.961	15.010.276	1.472.680.939	527.680	1.460.513	10.881.821	62.359	670.632	7.057.868	153.891	790.974	7.990.699
quickSort MidOfFirstMidLast Element	257.459	846.770	5.656.432	37.561	388.741	4.529.595	121.071	1.230.769	12.043.303	10.940	175.408	1.325.948
dualPivot QuickSort	4.482.188	849.322	4.590.130	1.769.026	20.194.093	1.279.475.003	322.370	15.426.732	1.525.905.590	228.649	14.800.225	1.530.282.388
introSort	4.739.646	2.228.513	5.216.637	38.656	104.296	1.282.917	39.019	124.353	989.721	9.481	143.316	1.005.401

Table 4.1.

6 Conclusion

Four algorithms were used in comparative analysis. The difference between them helps in determining the best usage of them. Furthermore, working on improve such sorting algorithm is needed in order to facilities how its applications work in more efficient way to reduce time & resources.

This paper presented the survey and comparison of different sorting algorithms along with the results of practical performance. The main findings of this study are that three factors such as running time, number of elements in sorted array and status of elements in sorted array that are critical to efficiency of sorting algorithms. Experiment results clearly show that the theoretical performance behavior is relevant to the practical performance of each sorting algorithm. However, there is a slight difference in the relative performance of algorithms in some test cases. In addition, the experiment results proved that algorithm’s behavior will change according to the size of elements to be sorted and type of input elements. Which conclude that each algorithm has its own advantage and disadvantage.

7 References

[1] Review on sorting algorithms A comparative study on two sorting algorithms By Pooja Adhikari.

[2] Comparison of Sorting Algorithms (On the Basis of Average Case) Pankaj Sareen.

[3] Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, fifth Indian printing (Prentice Hall of India private limited), New Delhi-110001

[4] Sharma, V. and Singh, S. (2008) Performance Study of Improved Heap Sort Algorithm and Other Sorting Algorithms on Different Platforms. IJCSNS International Journal of Computer Science and Network Security, VOL 8, No 4

[5] https://www.geeksforgeeks.org/heap-sort/

[6] https://www.baeldung.com/java-quicksort

[7] C. A. R. Hoare, "Quicksort," Computer Journal, 5, pp 10 – 15 1962.

[8] https://betterexplained.com/articles/sorting-algorithms/

[9] http://cooervo.github.io/Algorithms-DataStructures-BigONotation/index.html